Appeared in the Workshop on Parallel Execution of Sequential Programs on Multi-core Architectures (PESPMA), Beijing, China, June 2008, co-located with the 35th International Symposium on Computer Architecture (ISCA)

The Scalability of Scheduling Algorithms for Unpredictably Heterogeneous CMP Architectures

Jonathan A. Winter and David H. Albonesi
Computer Systems Laboratory, Cornell University
{winter, albonesi}@csl.cornell.edu

Abstract

Future chip multiprocessors (CMPs) will be expected to deliver robust performance in the face of manufacturing process variations and hard errors. While prior research has addressed how to keep these chips functional, these degraded CMPs may still be unusable because of lost performance and power efficiency. The unpredictable nature of the variability and faults will create dynamic heterogeneity among the cores of these CMPs. In prior work, we developed scheduling algorithms that mitigate the impact of core degradation by appropriately matching applications in the workload with cores of differing capabilities. In this paper, we study the scalability of these algorithms in CMPs ranging from four to sixty-four cores. Our results show that our algorithms which perform multiple exploration steps in parallel have greater potential to scale to larger CMP organizations.

1. Introduction

As Moore's Law continues to deliver exponentially more transistors on a die over time, computer architects have transitioned to the multi-core approach whereby these transistors are used to create additional cores on the same die. While this strategy does not improve individual single-threaded applications, it does permit higher throughput on multithreaded workloads, including those comprising multiple sequential applications. However, increasing microprocessor power dissipation and reliability challenges threaten to limit the benefits of further scaling.

While transient (soft) errors are a near-term research focus, permanent (hard) errors and circuit variability are projected to become a significant challenge [4]. In this work, we focus on permanent faults and variations caused by imperfections in chip manufacturing and lifetime wear-out. These manifest as inoperable transistors, open or shorted wires, slower critical timing paths (decreasing operating frequency), and higher leakage power. Since the errors and variations are a result of largely random physical processes that occur during manufacturing and usage, each core on a CMP will be uniquely affected. The result will be an unpredictably heterogeneous CMP, even though it may have been designed as a homogeneous architecture.

As these hard errors and variations become more prevalent, manufacturers will not be able to afford to discard affected chips, but will instead be forced to develop microprocessors that tolerate these shortcomings. Many researchers have developed resiliency approaches that allow processors to remain functional despite reliability problems [1,5,23,24,25,27,28]. However, these functional but degraded cores may fail to provide the minimum expected level of performance, or may dissipate unacceptably high amounts of power.

Our previous work [30] proposed a number of scheduling algorithms that recapture most of the lost performance and power efficiency by intelligently matching the application characteristics of a workload of sequential programs to the level of functionality of each degraded core. Two different approaches were taken to develop effective schedulers in [30]. The first approach is to make simplifying assumptions about the interactions between the sequential applications running on each core. This reduces the general scheduling problem to the Assignment Problem which we solve using a scheduler based on the Hungarian Algorithm [18]. The second set of scheduling algorithms is based around iterative optimization algorithms which are commonly applied to similar hard combinatorial problems [20,22]. Compared to the Hungarian scheduling algorithm, these iterative techniques are simpler to implement and less computationally intensive. The best algorithm is a variant of local search that explores the scheduling search space by swapping the core assignment of pairs of applications in every iteration.

Our prior work only considers an eight core CMP with a single degraded configuration. This paper explores the relative scalability of the different scheduling algorithms on CMPs with four to sixty-four cores, while considering multiple randomly generated degraded configurations, and multiple randomly generated workloads. An N core CMP has N! possible

scheduling assignments leading to a rapidly growing search space as the number of cores increases. We characterize the runtime of the Hungarian scheduling algorithm on different problem sizes to determine if the complexity of the algorithm is destined to restrict its usage to smaller CMPs. Moreover, we assess the significance of the number of exploration intervals employed by our iterative search algorithms. Through this analysis, we direct future research in scheduling algorithms for large-scale unpredictably heterogeneous CMPs towards algorithms that are most likely to scale well.

2. Overview of Scheduling Algorithms

We briefly describe our scheduling algorithms here and refer the reader to [30] for a more detailed explanation. All of our algorithms are designed to operate over fixed, relatively short periods. At the end of each period, the scheduling decisions are reassessed to account for application phase changes, program completion, and newly arriving applications. Our baseline algorithm rotates the application to core assignment in a round robin fashion each scheduling interval. The goal is to avoid worst case behavior by evenly assigning the application to each degraded core.

The more sophisticated schedulers start each period with an exploration phase during which the algorithm samples a number of possible assignments to gather profiling information. This information is then used to determine a predicted best schedule which is run during the steady phase. We study three algorithms that employ this two phase approach: the Hungarian scheduling algorithm, global search, and local search.

The Hungarian scheduling algorithm is based around the Hungarian Algorithm which solves a classic operations research problem called the Assignment Problem, also known as Weighted Bipartite Matching in graph theory [18]. This scheduler assumes that an application's sampled performance and power characteristics on a degraded core during the exploration phase will accurately reflect the program's behavior throughout the scheduling period and that this behavior will hold regardless of the core assignment of the other concurrently running applications. Hence, the algorithm assumes the applications do not exhibit dynamic phase behavior over the short scheduling period, and that interactions, such as contention for off-chip bandwidth and inter-core heating, are negligible between sequential programs running in parallel on the CMP. During the exploration phase, the algorithm samples each application on each core. Using the above assumptions, finding the best schedule based on this sampled information is reduced to applying the Hungarian Algorithm.

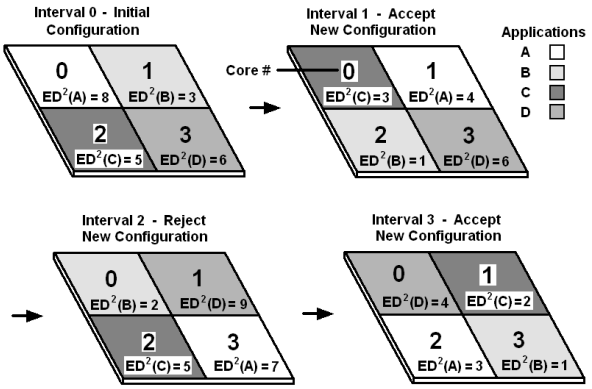

Figure 1: Global search algorithm

Global search and local search are iterative algorithms which greedily explore the space of possible schedules for an assignment that improves the best schedule found thus far. The exploration phase is divided into a number of evenly sized intervals and the algorithms sample a new configuration each time. Global search (Figure 1) simply samples a random configuration each time. The best performing configuration is then used during the steady phase.

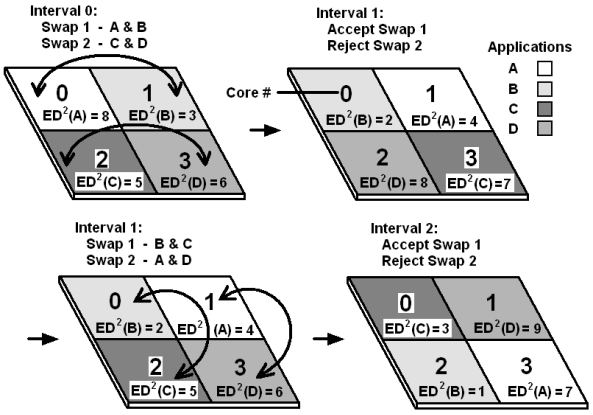

Figure 2: Two swap local search algorithm

Local search picks a random schedule to start and then repeatedly explores the neighborhood of the current best assignment for a better schedule. The neighborhood of the current schedule is all schedules that can be generated through a small change to the current schedule. In each subsequent iteration, one such neighbor is randomly sampled. In our implementation, the neighborhood is defined as those assignments that can be derived by performing a fixed number of pair-wise swaps to the original assignment. An example of local search with two swaps is illustrated in Figure 2.

In [30], we found that the more benchmarks swapped each interval, the better local search performed. In this paper, we study Local Search N/2 which performs N/2 pair-wise swaps involving all applications running on the cores. Our multi-swap variants of local search accept good swaps while discarding others. Performing multiple swaps and allowing the partial acceptance of swaps improves performance because it more rapidly traverses the search space. In this work, we study whether these algorithmic features continue to provide benefits beyond the single eight core degraded configuration employed in [30].

3. Methodology

Our simulation framework uses a hierarchical infrastructure to rapidly evaluate large-scale multi-core processors (Figure 3). In this study, we focus on sequential programs, in this case from the SPEC CPU2000 suite, and run a set of these applications simultaneously on our CMP, one on each core. With these sequential workloads, there is only limited direct interaction between concurrently executing programs. We assume private L2 caches which eliminates inter-thread cache effects, and that the L2 caches surround each core, which also significantly reduces inter-core heating that impacts leakage power. Finally, we assume that the off-chip bandwidth is statically partitioned among the cores, which avoids interaction through contention between accesses to main memory and I/O.

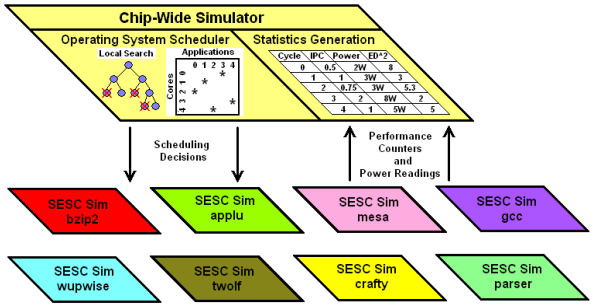

Figure 3: Multi-core simulation framework

With these assumptions, our framework evaluates a large-scale CMP by combining single application, single core simulations. We perform single core simulations of each application for each possible degraded core configuration using an improved version of the SESC simulator [21]. We augmented SESC's power and thermal modeling with Cacti 4.0 [29], an improved version of Wattch [6], the block model of Hotspot 3.0 [26], and an improved version of HotLeakage [31]. Our baseline core is a single-threaded, three-way superscalar, out-of-order processor. See [30] for more details about the core microarchitecture.

These single-core simulations are combined by a chip-wide simulator implemented in Perl which performs two roles. First, it compiles the performance, power, and thermal simulation statistics from SESC for each application/core pair into the complete statistics for the chip multiprocessor. Second, it performs the role of the operating system scheduler by implementing the various scheduling algorithms and executing the required single-core simulations for the exploration and steady phases of the chip simulation. This approach makes the simulation runtime relatively insensitive to the number of cores in the CMP configuration. Moreover, the execution results from running a particular application on a certain degraded core configuration can be reused in future CMP simulations whenever that application/core configuration reoccurs.

In this study, we evaluate a 40 million cycle window of each application execution. This window is used for each round robin rotation interval, each Hungarian scheduling algorithm sample, and each search algorithm exploration interval. The algorithms attempt to optimize the scheduling assignment that should be used for this interval. To evaluate the algorithms, a chip-wide simulator calculates the average energy-delay-squared (ED^2) product of the best schedule found by the algorithms based on the performance and power dissipation values from SESC. One exception is the round robin algorithm, where the mean performance and power values across each of the N rotations for an N core CMP are used to compute the ED^2 in order to account for the averaging effect of the rotation. The average ED^2 is compared against the average ED^2 of a baseline architecture with homogeneous cores without any errors or variations, as well as an oracle scheduler which uses the optimal assignment of applications to cores over the 40M cycle interval.

We examine chip multiprocessors with four, eight, 16, 32, and 64 cores. We randomly generate four appropriately sized workloads from among 17 SPEC CPU 2000 benchmarks for each CMP organization (The other SPEC benchmarks do not run on our SESC simulator). Moreover, we consider three types of degradation. We model the disabling of all or part of a processor component such as an ALU or a set of queue entries. Cores can also be restricted to a lower than nominal frequency by process variations. Finally, sections of the core can suffer from increased leakage causing higher than normal static power. Table 1 presents a list of the core degradations possible for each type of fault. To make the study more feasible, we

reduce the space of possible degraded configurations by assuming a core can be affected by only one problem (or none) from each category of degradation. In all, there are 12 × 4 × 7 = 336 such degraded core configurations. With 17 applications, and our use of the fixed 40M cycle intervals, we run a total of 5,712 simulations that are reused across the scalability study.

Table 1: Possible forms of core degradation

Category of Degradation	List of Options
Degraded Component	none
	memory latency is doubled
	half the L2 cache
	half the L1 Icache
	front-end bandwidth is reduced from 3-way to 2-way fetch/decode/rename
	half the integer issue queue
	integer issue bandwidth is reduced to one
	one integer ALU is disabled
	half the load queue
	half the store queue
	half the L1 Dcache
	half the re-order buffer
Frequency Degradation	none (4 GHz)
	10% (3.6 GHz)
	20% (3.2 GHz)
	30% (2.8 GHz)
Increased Leakage	none
	2X nominal in L1 caches and TLBs
	2X nominal in front-end and ROB
	2X nominal in integer back-end
	2X nominal in floating point back-end
	2X nominal in load and store queues
	2X nominal across core (excluding L2)

4. Results and Discussion

4.1. Scheduling Algorithm Performance

Our goal is to characterize the potential of our scheduling algorithms and gauge how this potential is affected by scaling the number of cores. Consequently, we initially assume that our algorithms have sufficient time to sample different points in the search space. For instance, the Hungarian scheduling algorithm is able to obtain all N^2 samples of the execution of each of the N applications on each of the N cores. For large-scale CMPs, there may not be enough time to obtain all the samples, unless the samples are very short, but then they may not reflect the long term behavior of the application.

In addition, we report the ED^2 results for the steady phase of the algorithms, ignoring the overhead of sub-optimally executing applications during the exploration phase, since it is unclear how long the exploration and steady phases should be. Future work will consider the impact of different scheduling period lengths and ratios of exploration time to steady phase time.

Figure 4 shows the overall results of running each of the scheduling algorithms as the number of cores in the CMP is varied from four to sixty-four. Each bar represents the average increase in ED^2 across 16 different simulations (four workloads run on each of four degraded CMP configurations) relative to a baseline CMP of the same size unaffected by hard errors or variations.

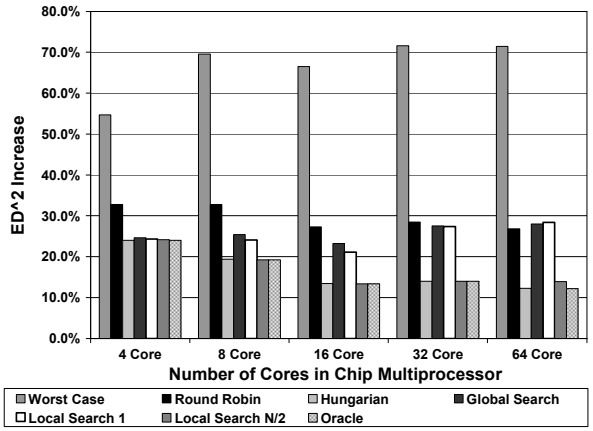

Figure 4: Overall scalability study results

The first bar in the graph represents the worst case assignment of application to cores, or the maximum possible loss in power/performance efficiency if programs were assigned to cores randomly without any thought towards core heterogeneity and the affinity of applications for some cores. For the four core designs, the worst case can be as bad as 55% above the baseline with no degradation, reaching 71% worse ED^2 for larger CMPs. The last bar in the graph represents an oracle scheduler that has complete knowledge, including the performance of the applications on the baseline core, and thus can find the scheduling assignment closest in ED^2 to the baseline.

From the results, we see that the naïve round robin algorithm performs quite poorly although much better than the worst case. This algorithm does quite well on homogeneous CMPs and on multi-core architectures which are designed intentionally to be heterogeneous. However, in our scenario round robin's averaging effect is not capable of reclaiming the full potential of

the baseline design, leading to energy-delay-squared increase of over 25% across all configurations.

The Hungarian scheduling algorithm finds very close to the optimal schedule, coming within 0.15% of the ED^2 of the oracle scheduler. The algorithm also reduces the impact of the core degradation with more cores. It fares best with 64 cores, reducing the ED^2 increase from 26.8% for round robin to only 12.3%, reclaiming more than half the lost efficiency.

For the search algorithms, we allow each algorithm to use 25 exploration intervals as in [30]. (Section 4.3 shows how the number of search intervals affects the success of the algorithm.) Figure 5 shows that while global search is competitive for smaller CMP configurations, it scales quite poorly, eventually lagging behind the Hungarian scheduling algorithm by 14.3% for 64 cores. As the search space increases with more cores, global search cannot perform well when evaluating only 25 randomly sampled schedules.

Local Search 1, which performs one swap to find a neighbor in each exploration interval, initially fairs better than global search but still fails to scale to larger CMPs. As the number of cores increases, the search space explodes and for 64 cores, it's better to run global search and evaluate 25 random configurations than to run local search in one corner of the search space.

Local Search N/2 swaps the core assignment of every application with another one in each search interval. With the ability to incorporate good swaps into the best solution and discard poor swaps, Local Search N/2 actually evaluates $2^{N/2}$ scheduling assignments each interval. This means that the algorithm evaluates more assignments as the number of cores grows in the same period of time. This feature may provide Local Search N/2 with the ability to scale to larger multi-core architectures. In our study, this search algorithm's ED^2 increase is never more than 1.7% above the oracle scheduler across all core sizes.

Another insight from this study is that as the number of cores is increased, the potential for the scheduling algorithms to mitigate performance losses and power dissipation gains from unreliable cores increases. While the oracle algorithm delivers an ED^2 24% higher than the baseline for a four core chip, it gets progressively better until it is only 12.2% worse for 64 cores. Likewise the oracle achieves an ED^2 8.8% less than round robin for four cores, but this improves to 14.6% for 64 cores. The reason that the oracle, Hungarian, and Local Search N/2 schedulers perform better relative to the baseline as the number of cores scales up is that the application and core configuration diversity increases with larger CMPs. With more programs and more degraded configurations, there is more of an opportunity to find a really good match between each application and core and reduce the performance and power penalty caused by the errors and variations.

4.2. Scalability of Hungarian Scheduler

In order to assess the runtime of the Hungarian Algorithm on different size chip multiprocessors, we took the algorithm for our chip-wide simulator and re-implemented it in C. We developed an $O(n^4)$ version of the algorithm which is easy to program and verify for correctness. The algorithm was then compiled and we simulated its execution on our version of SESC using the baseline core configuration. As input, we used the sampled data from each of the actual Hungarian scheduling algorithm runs from the study in Section 4.1.

Figure 5 shows the results. Each bar represents the average runtime for the 16 configurations used for each CMP size. While the runtime is negligible for the smaller configurations, it becomes quite substantial for a 64 core CMP, taking over 15 million cycles. The curved black line and the equation at the top of the graph represent the Microsoft Excel 4th order polynomial trendline. The R^2 value of 1.0 indicates a perfect fit to the data, consistent with the $O(n^4)$ complexity of our algorithm.

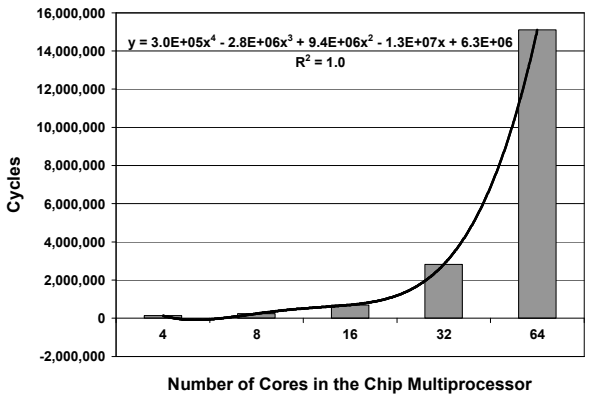

Figure 5: Runtime of Hungarian Algorithm

The rapidly growing cost of computing the solution to the Assignment Problem raises concerns about the scalability of the Hungarian scheduling algorithm to future large-scale CMPs. While a more efficient implementation can reduce the time complexity to $O(n^3)$ [18], this runtime may still be too long for practical use. Future work will consider applying parallelized version of the Hungarian Algorithm to reduce the computation cost [7]. The Hungarian scheduling algorithm also needs to collect N intervals

worth of sampled execution as input to the computation. This again poses a scalability problem for large CMPs. We plan to explore incremental variants of the Hungarian Algorithm that can resample only the applications that have changed phases or have recently entered the system. These algorithms can quickly modify the current best assignment to adjust for these changes, rather than recompute the entire assignment from scratch [17].

4.3. The Impact of the Number of Intervals on the Search Algorithms

Our final study seeks to evaluate how the number of intervals allocated to the search algorithms affects their performance. In Section 4.1, we allocated 25 intervals to each of the three search algorithms. In Figure 6, we show how these three algorithm perform as the number of intervals is varied from five to 100. The base segment of each bar in the graph shows the ED^2 increase of the oracle scheduler. This represents the lower bound on the performance of the search algorithms. The next segment shows how well the search algorithms do when given the maximum of 100 intervals to try different points in the search space. Each subsequent bar on top represents the added increase in ED^2 when the algorithms have fewer search intervals. For each of the search segments, the bar represents the results of 160 runs, corresponding to ten runs with different random seeds for each of the four application workloads and four degraded core configurations.

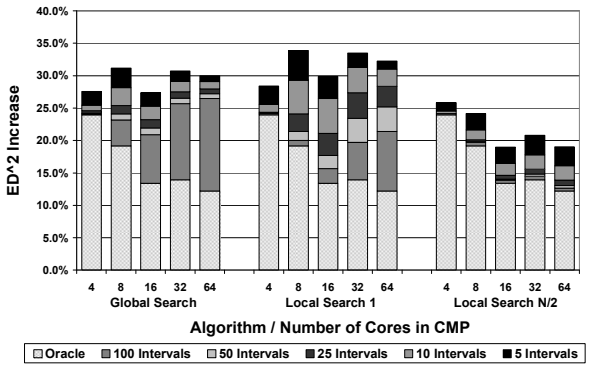

Figure 6: The impact of the number of intervals on the search effectiveness

Global search performs much worse than the oracle for CMP sizes greater than four cores even with 100 search intervals. The search space is simply too big to explore and more random samples are needed to guarantee good results.

When given 100 intervals, Local Search 1 does better than global search up to 16 cores. However, as the search space rapidly increases, the performance of Local Search 1 degrades. Furthermore, reducing the number of sampling intervals significantly increases its ED^2. With only five search intervals, Local Search 1 does even worse than global search because it explores a tiny segment of the space.

Local Search N/2 holds the most promise for scaling to larger CMPs. With 100 search intervals, its performance is never more than 0.6% above the oracle. Local Search N/2 continues to do well even with only 50 and 25 intervals, but the ED^2 starts to increase significantly when only ten and five search intervals are permitted. Local Search N/2 has two distinct advantages over the Hungarian scheduling algorithm. First, it only needs to perform small amounts of computation (O(n) compared to Hungarian's $O(n^3)$) to generate random swap pairs and evaluate if the swapped applications performed better or worse on their new core assignment. Second, by evaluating $2^{N/2}$ scheduling assignments in parallel every search interval, Local Search N/2 achieves good results with a shorter exploration phase than the Hungarian scheduler's N samples.

5. Related Work

A number of prior research areas closely relate to our degraded CMP scheduling problem. Previous work strives to understand, model, and mitigate manufacturing process variations (PV). Most work on PV focuses on the semiconductor device and circuit level, but a number of researchers have devised system-level approaches. Humenay et al. [9, 10] examine how parameter variations specifically impact multi-core chips. Several studies [15,16,19] propose to reduce the negative impact of variations on frequency and yield. Many of these mechanisms create heterogeneity on a CMP by disabling array elements or creating variable access times.

Other previous research uses the operating system to improve CMP energy efficiency. Juang et al. [12] argue for coordinated formal control-theoretic methods to manage energy efficiency in CMPs. Isci et al. [11] further develop globally aware policies to dynamically tune DVFS to workload characteristics to maximize performance under a chip-wide power constraint. While this effort has similar elements to ours, they use DVFS to improve efficiency, whereas in our heterogeneous CMP, we use core scheduling.

In designed heterogeneous CMPs [3,14], the heterogeneity is architected into the system rather than

the unplanned result of hardware faults and variations. As a result, the degree and nature of heterogeneity is quite different. Kumar et al. [14] focus on multi-programmed performance and applications are scheduled on cores to best match execution requirements. However, since only two types of cores are used, the solution space is small and thus a simple sampling scheme achieves good assignments. Becchi and Crowley [3] extend that work to use performance driven heuristics for scheduling. Our scheduling problem is far more complex: an unpredictably large number of heterogeneous organizations can arise in term of frequency, dynamic power, and leakage currents, in addition to architectural parameters.

Kumar et al. [13] study heterogeneous architectures where the cores are not restricted to a few configurations. The goal is to determine how much heterogeneity is necessary and how the cores should be designed to fit a given power budget. They focus on the design issues rather than the scheduling aspect. Balakrishnan et al. [2] study the impact of asymmetry in core frequency on parallel commercial workloads using a hardware prototype. Ghiasi et al. [8] examine heterogeneity due to cores running at different voltages and frequencies. While their work adapts the core voltages and frequencies, we investigate cores with unpredictably heterogeneous frequencies and leakage. We adapt the workload assignment to mitigate possible negative affects.

6. Conclusions

In this paper we explore the scalability of scheduling algorithms that we previously proposed for unpredictably heterogeneous CMPs. We find that global search and local search with one swap fail to scale effectively because they do not evaluate a sufficiently large portion of the search space during their exploration phases. The Hungarian scheduling algorithm achieves energy-delay-squared values extremely close to the oracle scheduler but suffers from the dual problems of high computational complexity and the need to collect a large number of samples during the exploration phase. Future work will examine variants of the Hungarian scheduler that address these scalability limiting factors. Local Search N/2 holds the most promise for scheduling in large-scale chip multiprocessors because it has a lower computational cost and is capable of assessing many application assignment options in parallel, making it effective even with few sampling intervals. Future work will develop schedulers specifically tailored to function well on CMPs with many cores, and when there is limited exploration time and dynamically changing workloads.

Acknowledgements

The authors would like to thank the anonymous reviewers for their useful feedback. This research is supported by NSF grants CCF-0732300 and CCF-0541321.

References

[1] N. Aggarwal, P. Ranganathan, N.P. Jouppi, and J.E. Smith. Configurable Isolation: Building High Availability Systems with Commodity Multi-Core Processors. *International Symposium on Computer Architecture*, 2007.

[2] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The Impact of Performance Asymmetry in Emerging Multicore Architectures. *International Symposium on Computer Architecture*, 2005.

[3] M. Becchi and P. Crowley. Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures. *ACM International Conference on Computing Frontiers*, 2006.

[4] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter Variations and Impact on Circuits and Microarchitecture. *Design Automation Conference*, 21.1, 2003.

[5] F.A. Bower, D.J. Sorin, and S. Ozev. A Mechanism for Online Diagnosis of Hard Faults in Microprocessors. *International Symposium on Microarchitecture*, 2005.

[6] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. *International Symposium on Computer Architecture,* 2000.

[7] M. Fayyazi, D. Kaeli, and W. Meleis. Parallel Maximum Weight Bipartite Matching for Scheduling in Input-Queued Switches. *International Parallel and Distributed Processing Symposium*, 2004.

[8] S. Ghiasi, T. Keller, and F. Rawson. Scheduling for Heterogeneous Processors in Server Systems. *ACM International Conference on Computing Frontiers*, 2005.

[9] E. Humenay, D. Tarjan, and K. Skadron. Impact of Parameter Variations on Multi-Core Chips. *Workshop on Architectural Support for Gigascale Integration*, 2006.

[10] E. Humenay, D. Tarjan, and K. Skadron. Impact of Process Variations on Multi-Core Performance Symmetry. *Design, Automation and Test in Europe*, 2007.

[11] C. Isci, A. Buyuktosunoglu, C-Y. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. *International Symposium on Microarchitecture*, 2006.

[12] P. Juang, Q. Wu, L-S. Peh, M. Martonosi, and D.W. Clark. Coordinated, Distributed, Formal Energy Management of CMP Multiprocessors. *International Symposium on Low Power Electronics and Design*, 2005.

[13] R. Kumar, D.M. Tullsen, and N.P. Jouppi. Core Architecture Optimization for Heterogeneous Chip Multiprocessors. *International Symposium on Parallel Architectures and Compilation Techniques*, 2006.

[14] R. Kumar, D.M. Tullsen, P. Ranganathan, N.P. Jouppi, and K.I. Farkas. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. *International Symposium on Computer Architecture*, 2004.

[15] X. Liang and D. Brooks. Microarchitecture Parameter Selection to Optimize System Performance Under Process Variation. *International Conference on Computer-Aided Design*, 2006.

[16] X. Liang and D. Brooks. Mitigating the Impact of Process Variations on Processor Register Files and Execution Units. *International Symposium on Microarchitecture*, 2006.

[17] G.A. Mills-Tettey, A. Stentz, and M.B. Dias. The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs. *Carnegie Mellon University, Robotics Institute, Technical Report CMU-RI-TR-07-27*, 2007.

[18] J. Munkres. Algorithms for Assignment and Transportation Problems. *Journal of the Society of Industrial and Applied Mathematics*. 5(1), 1957.

[19] S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou. Yield-Aware Cache Architectures. *International Symposium on Microarchitecture*, 2006.

[20] C.R. Reeves (Editor). *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill Book Company, London, UK, 1995.

[21] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. *SESC Simulator*. http://sesc.sourceforge.net, 2005.

[22] S.M. Sait and H. Youssef. *Iterative Computer Algorithms with Applications in Engineering*. IEEE Computer Society, Los Alamitos, CA, 1999.

[23] E. Schuchman and T.N. Vijaykumar. Rescue: A Microarchitecture for Testability and Defect Tolerance. *International Symposium on Computer Architecture*, 2005.

[24] P. Shivakumar, S.W. Keckler, CR. Moore, and D. Burger. Exploiting Microarchitectural Redundancy for Defect Tolerance. *International Conference on Computer Design*, 2003.

[25] S. Shyam, K. Constantinides, S. Phadke, V. Bertacco, and T. Austin. Ultra Low-Cost Defect Protection for Microprocessor Pipelines. *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[26] K. Skadron, M.R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture. *International Symposium on Computer Architecture*, 2003.

[27] J. Srinivasan, S.V. Adve, P. Bose, and J.A. Rivers. The Case for Lifetime Reliability-Aware Microprocessors. *International Symposium on Computer Architecture*, 2004.

[28] J. Srinivasan, S.V. Adve, P. Bose, and J.A. Rivers. Exploiting Structural Duplication for Lifetime Reliability Enhancement. *International Symposium on Computer Architecture*, 2005.

[29] D. Tarjan, S. Thoziyoor, and N.P. Jouppi. CACTI 4.0. *HP Laboratories Technical Report HPL-2006-86*, 2006.

[30] J.A. Winter and D.H. Albonesi. Scheduling Algorithms for Unpredictably Heterogeneous CMP Architectures. *International Conference on Dependable Systems and Networks*, 2008.

[31] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects. *The University of Virginia, Department of Computer Science, Technical Report CS-2003-05*, 2003.